Our Ref.: 41016.P004

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

Cell Based Data Processing

Inventor(s):
Adam Bosworth
David Bau
Eric Vasilik

Prepared by:

Columbia IP Law Group, LLC
Seattle/Kirkland Office

"Express Mail" label number: EL743034045US

Cell Based Data Processing

Related Applications

This non-provisional application is related to and claims priority to provisional application number 60/246,915, entitled "A Data Processing Method Employing Cell Based Data Flow Description", filed on November 10, 2000, which is hereby fully incorporated by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of data processing. More specifically, the present invention relates data processing specification and execution.

2. Background Information

Ever since the invention of the first computer, computer scientists have continuously tried to improve the productivity of programmers, such that more applications can be developed to take advantage of the continuous advancements being made in the art of computer and related technologies. First assembler languages were developed to replace machine languages. Then, high level languages, such as FORTRAN, COBOL, PL/I and so forth, were developed to further improve the productivity of programmers. Development of high level languages were followed by structured languages such as Pascal and C, and then object oriented programming languages such as C++. To facilitate development of

the Internet and the World Wide Web, "new" languages such as the Hypertext Markup Language (HTML), Java, Javascript, Perl and CGI were developed.

While great strides had been made in the past decades, advancements in integrated circuit, computer architecture, telecommunication and networking technology continue to outpace the productivity improvement of the programming community. Application development remains substantially a bottleneck to the introduction and application of the latest computer and related technology advancements. Notwithstanding the development of "power user" type of application development languages/facilities, such as Visual Basic, and the continuing improvement and extension to the more traditional languages, a need still exist for a more user friendly way in development data processing applications, for the more average, non-advanced users.

SUMMARY OF THE INVENTION

In accordance with the present invention, a data processing program is specified by way of a specification having a number of cell specifications specifying a number of data processing cells, with each data processing cell having a formula specifying an action or a computation. A cell may have one or more attributes referencing other cells. A cell formula may also reference a value of another cell or be executed conditionally.

In one embodiment, one of the cells is reserved as an output cell specifying the output, and a mnemonic is reserved for providing input to the specified data processing.

In one embodiment, an execution analyzer is provided to analyze the data processing specification, and identify the execution order of the cells. Further, an execution engine is provided to effectuate the specified data processing by executing the specified actions/computations of the cells in accordance with the determined execution order.

BRIEF DESCRIPTION OF DRAWINGS

The present invention will be described by way of exemplary embodiments, but not limitations, illustrated in the accompanying drawings in which like references denote similar elements, and in which:

Figure 1 illustrates an overview of the x-sheet data processing specification of the present invention, including the x-sheet execution analyzer and the x-sheet execution engine of the present invention, in accordance with one embodiment;

Figure 2 illustrates the relevant operational flow of the x-sheet execution analyzer of **Fig. 1**, in accordance with one embodiment;

Figures 3a-3b illustrate a graphical representation of an example execution flow, and an example data structure suitable for use to represent the example execution flow;

Figure 4 illustrates the relevant operational flow of the x-sheet execution engine of **Fig. 1**, in accordance with one embodiment; and

Figure 5 illustrates a computer system suitable for use to practice the present invention, in accordance with one embodiment.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, various aspects of the present invention will be described. However, it will be apparent to those skilled in the art that the present invention may be practiced with only some or all aspects of the present invention. For purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will also be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known features are omitted or simplified in order not to obscure the present invention.

Parts of the description will be presented in terms of operations performed by a computer system, using terms such as data, values, tags, references, and the like, consistent with the manner commonly employed by those skilled in the art to convey the substance of their work to others skilled in the art. As well understood by those skilled in the art, these quantities take the form of electrical, magnetic, or optical signals capable of being stored, transferred, combined, and otherwise manipulated through mechanical and electrical components of the computer system; and the term computer system include general purpose as well as special purpose data processing machines, systems, and the like, that are standalone, adjunct or embedded.

Various operations will be described as multiple discrete steps in turn, in a manner that is most helpful in understanding the present invention, however, the order of description should not be construed as to imply that these operations are necessarily order dependent. In particular, these operations need not be performed in the order of presentation.

Overview

Referring now **Figur 1**, wherein a block diagram illustrating an overview of the x-sheet data processing specification of the present invention, including a x-sheet execution analyzer and a x-sheet execution engine of the present invention, in accordance with one embodiment. In accordance with the present invention, x-sheets **102** (pronounced "cross sheets") are advantageously employed to specify data processing programs. As illustrated, each x-sheet **102** includes a number of x-cells **104** (pronounced "cross cells"), with each x-cell **104** including one or more formulas **110**, and each formula **110** specifying an action or a computation to be performed (when the x-cell is executed). Each x-cell **104** may include one or more attributes **108** referencing other x-cells **104**. Similarly, each formula **110** may also reference values of other x-cells **104**. In other words, from at least the execution point of view, x-cells **104** may be inter-dependent or interlocked with one other, thus the name "x-cell" and "x-sheet".

As illustrated, for the embodiment, a x-sheet execution analyzer **122** is also advantageously provided to analyze the x-sheets **102**, in particular, determining the execution flow of their x-cells **104**, "documenting" the flows in execution flow descriptions **132**. Further, a x-sheet execution engine **124** is provided to execute the x-cells **104** in accordance with the determined execution flow.

As a result, data processing operations may be advantageously specified and effectuated in a much more user friendly manner. These elements, x-sheet **102**, x-cell **104**, execution analyzer **122**, execution flow description **132**, execution engine **124**, and the manner they relate, interact and/or cooperate with each, will be described in further detail in turn below.

X-Sheet and X-Cells

Turning now first to x-sheets **102** and x-cells **104** of the present invention, and still referring to **Fig. 1**, as described earlier, each x-sheet **102** is employed to specify a data processing program, including a number of x-cells **104**, with each x-cell **104** specifying an action or a computation to be performed. Through their references to each other, via their attributes or their formulas, x-cells **104** are inter-dependent or interlocked with one other.

In one embodiment, x-cells **104** are delineated by beginning and ending x-cell tags, such as <x:xcell> and </x:xcell>, similar to tags employed by HTML and XML data structures (for familiarity purpose). Further, each x-cell **104** is uniquely named using a "name" attribute. One of the x-cells **104** is reserved as the output cell for outputting the result or results of the specified data processing. Consider the following example x-sheet,

```
<x:xsheet>
    <x:xcell name="preferences">
        <mydata>
            <favoritecolor>red</favoritecolor>
            <favoritetoy>ballon</favoritetoy>
        </mydata>
    </x:xcell>
    <x:output>
        <x: value-of select="$preferences/mydata/favoritecolor"/>
        <x: value-of select="$preferences/mydata/favoritetoy"/>
    </x:output>
</x:xsheet>
```

In the above example, the example x-sheet includes two x-cells. The first x-cell is named "preferences", whereas the second is the reserved "output" x-cell. X-cell "preferences" includes two formulas, one specifying a constant, "red" (as the favorite color), and the other specifying a constant "balloon" (as the favorite toy). The output x-cell also includes two formulas, specifying two output actions and referencing the values of x-cell "preferences" (i.e. the favorite color constant and the favorite toy constant). Accordingly, when executed, x-cell "preferences" is executed first, creating the constant values "red" and "balloon", and then the output x-cell is executed, outputting the string "red balloon". [The use of a formula to reference values of other x-cells, and the meaning of the expressions "value of" as well as "select" will be further described later.]

As described earlier, in addition to the formulas **110** of x-cells **104** being able to reference values of the x-cells **104**, the x-cells **104** themselves, via attributes **108**, may also reference the other x-cells. More specifically, a special "use" attribute is reserved for such purpose. Consider the following example x-sheet,

```
<x:xsheet>
    <x:xcell name="calculate" uses="$action $setup">
        <something/>
    </x:xcell>
    <x:xcell name="action" uses="$init">
        <another/>
    </x:xcell>
    <x:xcell name="setup">
        <x:value-of select="$init/yetanother"/>
    </x:xcell>
    <x:xcell name="init">
```

```
        <yetanother/>
    </x:xcell>
    etc.
</x:xsheet>
```

In this example, the "setup" x-cell refers to the "init" x-cell, and the "calculate" x-cell, via its "uses" attributes, refers to the "action" and "setup" x-cells. Accordingly, the "setup" x-cell" will be executed after the "init" x-cell. Similarly, the "calculate" x-cell with be executed after "setup" x-cell as well as the "action" x-cell. [The relative order between the "action" x-cell and the "setup" x-cell is considered "undefined".]

Additionally, in support of development of Internet applications, an x-cell, the "header" x-cell, is reserved for the specification of the "meta data", such as defining Java functions used by other x-cells, defining caching policies for the data processing specification, defining user authentication information, editing state, and so forth. An example "header" x-cell may be specified as follows:

```
<x:header name="coolsheet">
    <lastmod>August 19, 2000</lastmod>
    <editstate>
        <cursorpos>4</cursorpo>
        <windowsize>
            <width>1432</lwidth>
            <height>323</height>
        </windowsize>
    </editstate>
</x:header>
```

Further, a "process-content" attribute, set to either "true/false", is supported to facilitate specification of the manner in which an HTTP request is to be processed.

Such a request may be received when a x-sheet is hosted as a servlet. In one embodiment, when the attribute is not specified or set to "false", a HTTP request is read and converted to XML on behalf of the x-sheet. However, if the "process-content" attribute is set to "true", the input is left in the CGU-style format, allowing the x-sheet to read the body of the HTTP request itself.

In summary, x-cell elements are children of a x-sheet element. Each x-cell element has a *name* attribute uniquely naming the x-cell. The names "output" and "header" are reserved. Each x-cell may also have one or more attributes, including a *use* attribute referencing other x-cells.

X-Cell Formulas

Turning now to x-cell formulas **110** of the present invention, and still referring to **Fig.** 1, as described earlier, each x-cell formula **110** is employed to specify an action or computation. As illustrated, at least one mnemonic (e.g. $input) **112** is reserved for providing input to the specified data processing. Further, a number of "operator" elements are supported to facilitate specification of the actions or computations. In one embodiment, the operator elements include:

- select
- value of
- content of
- copy of
- if
- for

The "select" element

Each "select" element has a path, and is used to select a portion of an inner value of the path. As illustrated in some of the earlier examples, the path may point to another x-cell. Consider the following example

```
<x:output>
    The authority is: <myfunc:currentuser x:select="user/authority"/>
</x:output>
```

In this example, assuming "user" is an XML record with "authority" set to "supervisor", the output of the example would be

The authority is: supervisor

The "value of" element

Each "value-of" element also has a path, and is used to produce text results from the path specified by the *select* attribute. As also illustrated in some of the earlier examples, the path may point to another x-cell. Consider the following example

```
<x:value-of select="$input/parameters/record"/>
```

In this example, assuming also "parameter" is an XML record with "record" having "first" and "last" elements set to "John" and "Doe", the output of the example would be

JohnDoe.

The "content of" element

Each "content-of" element also has a path, and is used to produce XML results from the path specified by the *select* attribute. Similar to the *value-of* and *select* attributes, the path may point to another x-cell. Consider the earlier example again

```
<x:content-of select="$input/parameters/record"/>
```

Assuming again "parameter" is an XML record with "record" having "first" and "last" elements set to "John" and "Doe", the output of the example would be

<first>John</first>

<last>Doe</lase>.

In other words, the difference between "value-of" and "content-of" is that in the earlier case, the delimiters or tags are removed, whereas in the later case, they are not removed.

The "copy of" element

Each "copy-of" element also has a path, and is used to produce a node set from the path specified by the *select* attribute. Also similar to the earlier described attributes, the path may point to another x-cell. Consider the example

<x:copy-of select="$input/parameters/record"/>

In this example, all the records delineated between the "parameter" tags, i.e. <parameter> ... </parameter> are output, which may be

<record>

<first>John</first>

<last>Doe</last>.

</record>

<record>

<first>Jane</first>

<last>Doe</last>.

</record>

assuming these "records" are contents of the input "parameters".

The "if" element

The "if" element is used to perform a single Boolean test, causing either a <x:then> or a <x:else> section to be executed depending on the result of the test

(naturally, the <x:then> section is executed if the test is true, and the <x:else> section is executed if the test is false). Consider the following example

```
<x:if>
    <x:test x:select="./actual = ./submitted">
        <actual><x:value-of select="$realpassword"/></actual>
        <submitted><x:value-of select="$typedpassword"/></submitted>
    </x:test>
    <x:then>
        <message>Right password! Welcome to the secret area.</message>
        <result>ok</result>
    </x:then>
    <x:else>
        <message>Sorry! Only members allowed.</message>
        <result>reject</result>
    </x:else>
</x:if>
```

In this example, the contents of the <x:test> section is evaluated. The result is a document fragment containing the elements <actual> and <submitted> with some values inside them. The local path in the "x:select" attribute on the <x:text> element is executed in the context of the document fragment result of the test. The result of the path is casted to a path Boolean. Lastly, if the Boolean result was true, the <x:then> section is eevaluated and its contents become the value of the <x:if>. If the Boolean result was false, the <x:else> section is evaluated and its contents become the value of the <x:if>.

The "for" element

The "for" element is used to facilitate iteration over a list of nodes. Consider the following example

```
<x:for var="rec">
    <x:each x:select="record">
        <record><first>John</first><last>Doe</last></record>
        <record><first>Jane</first><last>Doe</last></record>
    </x:each>
    <x:do>
        <log:output>
          <message>Hello <x:value-of select="$rec/first"/></message>
        </log:output >
    </x:do>
</x:for>
```

In this example, the contents of the <x:eah> section is evaluated first. If a "x:select" attribute is present, it is evaluated and its result is interpreted as a node *list* (i.e. not a single node). In this case, the node list has two nodes: the two <record> elements. The name "$rec" is bound to each of the nodes in the list, beginning with the first. If the name shadows other name that is in scope, it is an error. That is, if there is a <x:xcell> names "rec" or and out <x:for> using a variable called "rec", an error will be signaled. If no error, for each of nodes in the list, the contents of the <x:do> section is evaluated once. The value of the <x:for> if the document fragment containing the concatenated values of all the evaluated <x:do> sections. Outside the <x:for>, the variable <$rec> is meaningless, and any reference would be signaled as an error.

X-sheet Execution Analyzer

As described earlier, a x-sheet execution analyzer is provided to parse and analyze an x-sheet to determine the execution flow of the x-cells. **Fig. 2** illustrates the operational flow of the relevant aspects of x-sheet execution analyzer **122** in accordance with one embodiment, whereas **Figs. 3a-3b** illustrate a graphical representation of an example execution flow, and an example data structure for representing the execution flow.

As illustrated, upon invocation, i.e. provided with an x-sheet for analysis, analyzer **122** would locate the next cell, block **202**. Recall that in one embodiment, each x-cell is delineated by beginning and ending x-cell tags. Locating these tags may be accomplished using anyone of a number of parsing techniques known in the compiler art. Upon locating the next x-cell, analyzer **122** would determine if the located x-cell references other x-cells, either by way of the "use" attribute, or by virtue of its formulas, block **204**. Similarly, detection of the present of certain attributes and syntactical elements may be accomplished using any syntax analysis techniques known in the compiler art.

Next, for the illustrated embodiment, upon determining the "interdependency" of the x-cell being analyzed with other x-cells, the interdependency information are output, block **206**. In one embodiment, the interdependency information are maintained by way of a directed graph (logically speaking) [see e.g. **Fig. 3a.**]. The data associated with the nodes and arcs of the logical graphic representations may be stored in any one of a number of suitable data structures known in the art, e.g. the tabular data structure illustrated in **Fig. 3b**.

Thereafter, analyzer **122** determines if additional x-cells are present and to be analyzed, block **208**. If additional x-cells are present and to be analyzed, the process continues back at block **202**. On the other hand, if all x-cells have been

analyzed, the cumulated interdependency information are ordered, block **210**, and then output as execution flow **132**, block **212**.

X-sheet Execution Engine

As described earlier, a x-sheet execution engine is provided to execute the x-sheets in accordance with their determined execution flows. **Fig. 4** illustrates the operational flow of the relevant aspects of x-sheet execution engine **124** in accordance with one embodiment.

As illustrated, upon invocation, i.e. provided with an analyzed x-sheet for execution, execution engine **124** would locate the first cell to be evaluated, as described by execution flow **132**, block **402**. Upon identifying the first x-cell to be evaluated, execution engine **124** proceeds to evaluate or facilitate to have the formulas evaluated, block **404**. The formulas are evaluated in accordance with the semantic meaning of the formula elements (i.e. *x:select, x:value-of, x:content-of, x:copy-of and so forth*), as described above. Evaluation of the these supported elements in accordance with their semantic meanings may similarly be accomplished using any one of a number of techniques known in the art for executing like kinds of elements in other languages.

After evaluating the first x-cell, execution engine **124** proceeds to determine if additional x-cells are to be executed, again in accordance with execution flow **132**, block **406**. If additional x-cells are to be executed, execution engine **124** "loads" the next x-cell for evaluations, block **408**. From block **408**, the process returns back to block **404**. On the other hand, if all x-cells have been evaluated, the process terminates.

Example Computer System

Figur 5 illustrates a computer system suitable for use to practice the present invention, in accordance with one embodiment. As shown, computer system **500** includes one or more processors **502** and system memory **504**. Additionally, computer system **500** includes mass storage devices **506** (such as diskette, hard drive, CDROM and so forth), input/output devices **508** (such as keyboard, cursor control and so forth) and communication interfaces **510** (such as network interface cards, modems and so forth). The elements are coupled to each other via system bus **512**, which represents one or more buses. In the case of multiple buses, they are bridged by one or more bus bridges (not shown). Each of these elements performs its conventional functions known in the art. In particular, system memory **504** and mass storage **506** are employed to store a working copy and a permanent copy of the programming instructions implementing the x-sheet data processing specifications, and their execution analyzer and engine. The permanent copy of the programming instructions may be loaded into mass storage **506** in the factory, or in the field, as described earlier, through a distribution medium (not shown) or through communication interface **510** (from a distribution server (not shown). The constitution of these elements **502-512** are known, and accordingly will not be further described.

Conclusion and Epilogue

Thus, it can be seen from the above descriptions, a novel method and apparatus for specifying data processing, and effectuating the specified data processing have been described. While the present invention has been described in terms of the above illustrated embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The present invention can

be practiced with modification and alteration within the spirit and scope of the appended claims. For examples, the present invention may be practiced with or without reserved output cells, input mnemonics, etc. The description is thus to be regarded as illustrative instead of restrictive on the present invention.

5